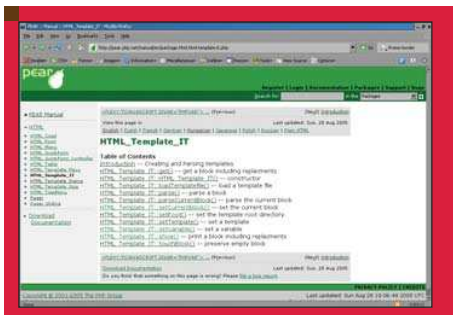
Sablonkezelés PHP nyelven, PEAR módra

Tovább folytatjuk a PEAR modulokból történő alkalmazásépítést. Legutóbb a DB adatbázis elvonatkoztatási réteget tanulmányoztuk részletesen, ezúttal a nem kevésbé fontos sablonkezelési témát boncolgatjuk, a legnépszerűbb témába vágó PEAR modul, az IT ismertetésével egybekötve.

Bármilyen programozási nyelvről legyen is szó, a kód tisztasága, átláthatósága kulcsfontosságú. Nem csak a továbbfejlesztésről van szó, de az esetleges hibakeresés is rémálommá válhat abban az esetben, ha nincs szerkezete a forráskódunknak. A cél elérésének módja alapvetően az, hogy a fejlesztő rendesen dolgozik, odafigyel, hogy milyen munkát ad ki a kezéből. Nem tud viszont mit kezdeni azzal a ténnyel, hogy számtalan szkriptnyelv, többek között a *PHP* is alapértelmezetten a *HTML* kódba épül be, tehát a kinézet és a kód összekeveredik, ami nem csak logikailag és a működés szempontjából nézve számít szerencsétlen körülménynek. A fenti állapot spagettikóddal összemérhető igénytelenségű kódot eredményez, amelyet a legjobb szándékkal sem lehet tisztán tartani. Az egyetlen megoldás a kód és a kinézet szétválasztása, elkülönítése.



Ez minden esetben valamilyen sablonkezelő rendszert használatával érhető el, amely a kinézetet tartalmazó sablont összeolvasztja az adatokat elkészítő *PHP* kimenettel, és az egészet kiteszi a felhasználónak. Így nem pusztán kódot és kinézetet lehet szétválasztani, de megvalósítható a manapság oly divatos *MVC* architektúra is, ahol a modell a nézet és a vezérlő jól elkülönített hármasából áll össze az alkalmazás.

A sablonkezelést valahogy úgy kell elképzelni, mint az előre megírt papír formanyomtatványokat, ahol ezen a bizonyos „okmányon” szerepelnek azok a részek, amelyek minden esetben ugyanúgy néznek ki, s nekünk az a dolgunk, hogy a változó adatokat kézzel töltsük ki. A programozás területén a formanyomtatvány szerepét a program megjelenési felülete tölti be, ahol szintén az a dolgunk, hogy a fix részek közé a program futása során beírjuk a változó adatokat, magyarul „ki kell töltenünk” a sablont.

Ehhez a sablonban meghatározott alakú címkéket helyezünk el, amelyeket aztán a program segítségével kicserélünk, átalakítunk, vagy épp a sablonban található logika szerint értelmezünk. Minden program esetében igaz az, hogy a futás egy része kizárólag a kinézet előállítására fordítódik. A sablonkezelő rendszerek használata során sincs ez másként, azzal a jelentős különbséggel, hogy elhatárolódik egymástól a működési és a megjelenítési logika. A gyakorlatban a teljes program vagy oldal egyetlen *PHP* címkéből áll, amely sablonkezelő rendszert utasítva írja ki a megjelenítendő *HTML* kódot az alapértelmezett kimenetre.

A sablonok és a PEAR

Mielőtt ismét újra feltalálnánk a kereket, érdemes egy kicsit körülnézni. Azt már tudjuk, hogy a *PEAR* rendszer épp az ilyen általános megoldások kezelésére nyújt kész megoldásokat, kezdjük hát itt a nézelődést.

A sablonkezelőket a *HTML* kategóriában, a *Template* csoportban találjuk. (Megjegyzés: a kategóriában nincsenek valódi csoportok, a sablonkezelőket is csak a nevük kezdete teszi csoporttá.) Jelenleg öt sablonkezelő csomag áll a fejlesztők rendelkezésére, mindegyikük végleges (stable) állapotban. Az összes sablonkezelőt azonban nehéz lenne bemutatni. Általában a végleges állapotúakat szoktam ismertetni, jelen esetben ez azonban nem szűkíti a kört, úgyhogy a másik jellemző módszert választottam: népszerűségük alapján döntöttem, s a statisztikák szerinti legkedveltebb csomagot választottam (sok tízezer ember nem tévedhet). Ez az *Integrated Templates* nevű (teljes nevén *HTML_Templates_IT*) *PEAR* összetevő, amely egy igen egyszerű és jól használható, csere alapú sablonrendszer.

Az IT csomag

Az *IT* csomag egy egyszerű sablonkezelő alkalmazásfejlesztési felület (*API*), amely döntően a benne elhelyezett különleges formátumú címkék értékekre történő kicserélését végzi. Egy az *IT* számára értelmezhető sablon a hagyományos *HTML* címkékből, {címke} alakú ún. helyfenntartókból (placeholder), és az ezeket tagoló blokkokból áll. Amikor a sablont „kitöltjük”, a {címke} alakú helyfenntartókat cserélgetjük ki a programból az általunk meghatáro-

1. lista Az első sablon

A sablon

```
<html>
  <body>
    <!– BEGIN header –>
      Üdvözöllek! Ma {DATE} van.
      <hr>
    <!– END  header –>

    <!– BEGIN footer –>
      <hr>
      Látogató IP-je: {IP}
    <!– END  footer –>
  </body>
</html>
```

A PHP forrás

```
<?php
  require_once "HTML/Template/IT.php";
  $tpl = new HTML_Template_IT("./templates");

  //sablon betöltése
  $tpl->loadTemplatefile("main.tpl.html");

  //aktuális blokk kijelölése
  $tpl->setCurrentBlock("header");
  //címke-érték megfeleltetés
  $tpl->setVariable("DATE",date("Y.m.d"));
  //az aktuális blokk lefordítása
  $tpl->parseCurrentBlock();

  //az oldal megjelenítése
  $tpl->show();
?>
```

zott értékekre. A hatékony működés érdekében ezek a címkék blokkokba vannak szervezve. Ha nem adunk meg ilyet, akkor is az alapértelmezett `__global` blokkban szerepelnek, csak legfeljebb a mindennapos használat során ez nem tűnik fel. Amikor címkecserét hajtunk végre, először kijelöljük, hogy mely blokkban szeretnénk mindezt megtenni, majd a `címke -> érték` hozzárendelés után a sablonkezelő értelmezi és „lefordítja” a kívánságunkat igazi ***HTML*** kódra.
Csak ezután fog az adott blokk megjelenni a kimeneten, ha majd arra utasítást adunk. Egy-egy blokk kitöltése, értelmezése (fordítása) tetszőleges számban következhet egymás után, így ismétlődő részeket (például) táblázatok sorait hozhatunk létre egyedüli sorokból. Ez így elsőre talán nehezen érthető, úgyhogy forduljunk a már jól megszokott módszerhez: lássunk egy példát!

Építsünk bemutató holnapot az IT segítségével!

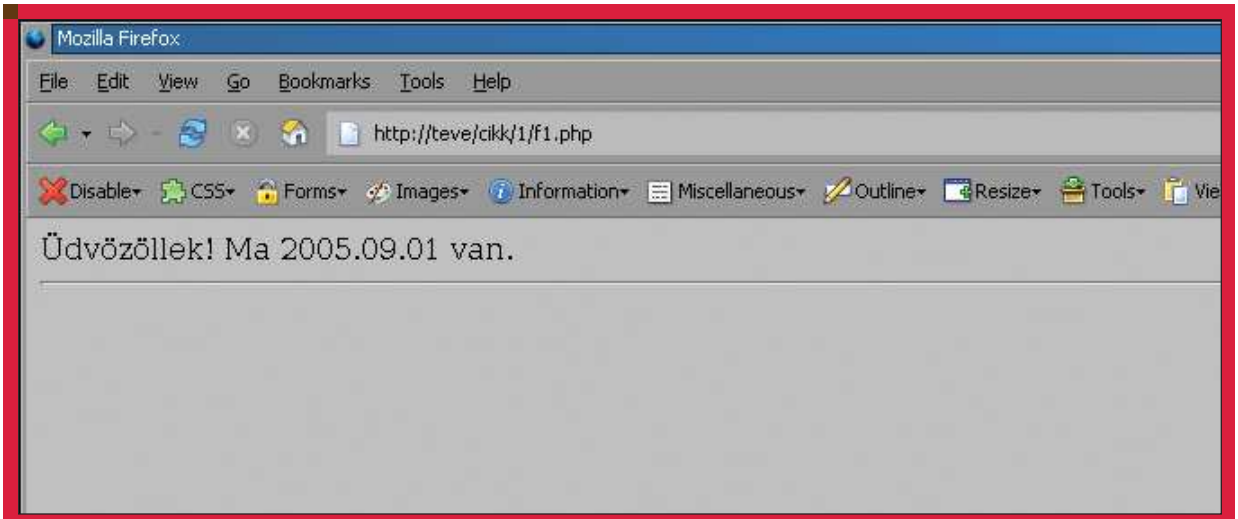
Esetünkben mindjárt példák egész sorával találkozhatunk. A feladat: apróbb lépésenként felépíteni egy nagyon-nagyon egyszerű, Forma-1-es világbajnokok adatainak megjelenítésére szolgáló webalkalmazást, amelynek építése során végigmenyünk az összes gyakori problémán. Előbb azonban telepítsük gépünkre a PEAR környezeten belül az ***IT***-t a

```
pear install HTML_Template_IT
```

parancs kiadásával.

A kezdeti tipegések

Az első lépés egy oldal létrehozása, amelynek van egy fejléce. Ez tartalmazza az aktuális dátumot. A sablonok az aktuális könyvtár ***templates*** alkönyvtárában találhatók.
A kód a sablonkezelő rendszer beemelésével kezdődik, rögtön ezután létre is hozunk egy példányt a sablonkezelőnkből, a továbbiakban ez a változó fogja megtestesíteni a rendszert, amely a paraméterként átadott könyvtárban fogja keresni a sablonokat.



1. ábra Az eredmény

2. lista A kiegészített sablon

A sablon

```
<html>
  <body>
    <!- BEGIN header ->
      Üdvözöllek! Ma {DATUM} van.
      <hr>
    <!- END  header ->
    <!- BEGIN footer ->
      <hr>
      Látogató IP-je: {IP}
    <!- END  footer ->
  </body>
</html>
```

A PHP forrás

```
<?php
  require_once "HTML/Template/IT.php";

  $tpl = new HTML_Template_IT("./templates");

  $tpl->loadTemplatefile("main.tpl.html");

  $tpl->setCurrentBlock("header");
  $tpl->setVariable("DATUM",date("Y.m.d"));
  $tpl->parseCurrentBlock();

  $tpl->setCurrentBlock("footer");
  $tpl->setVariable("IP",$_SERVER
  ➥['REMOTE_ADDR']);
  $tpl->parseCurrentBlock();

  $tpl->show();
?>
```

```
$tpl->loadTemplatefile("main.tpl.html");
```

Betölti a szükséges sablonfájlt, amely tartalmazza a blokkokat és címkéket, amiken később a műveleteket elvégezzük.

```
$tpl->setCurrentBlock("header");
```

Ha már betöltöttük a sablont, rendeljünk értékeket az elhelyezett címkékhez, ám mielőtt ezt megtennénk, ki kell jelölni, hogy melyik blokkban szeretnénk ezt megtenni. Egy változónév egy blokkban is szerepelhet többször, de azok azonos értékekre fognak kicserélődni. Ez a változás azonban nem érinti a más blokkokban elhelyezett azonos nevű változókat.

```
$tpl->setVariable("DATE",date("Y.m.d"));
```

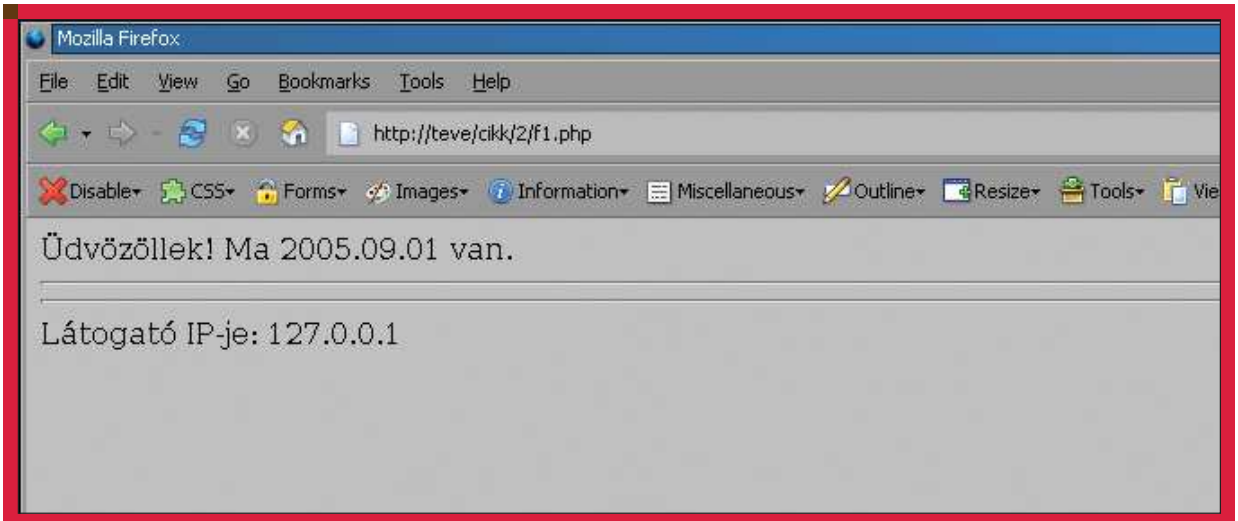
Ez a parancs végzi a konkrét megfeleltetést, és a `DATE` címkéhez a mai dátumot rendeli (természetesen csak a `header` blokkon belül).

```
$tpl->parseCurrentBlock();
```

A megfeleltetés után ezzel a paranccsal tudjuk kicseréltetni a címkéket értékekre, valamint ennek hatására az aktuális blokk bekerül a megjelenítendő tartalmak közé. Ellenkező esetben, ha egy blokkal nem csinálunk semmit, azaz nem rendelünk értékeket a címkékhez, és nem alkalmazzuk ezeket a változtatásokat, akkor az adott blokk tartalma nem fog megjelenni a kimeneten.

```
$tpl->show();
```

Ez a parancs végül kiteszi a már elkészült, kitöltött sablont az alapértelmezett kimenetre. Előfordulhat azonban, hogy nekünk nem erre van szükségünk, nem kiíratni szeretnénk az értéket, hanem a tartalmával műveletet végezni (levélben küldeni, stb.). Ilyen esetekben a `get()` metódust hasz-



2. ábra Az új eredmény

3. lista Ismétlődő elemek egy sablonban

A sablon

```
<html>
  <body>
    <!– BEGIN header –>
      Üdvözöllek! Ma {DATE}. van.
      <hr>
    <!– END  header –>

      <!– BEGIN list –>
        <h4>Versenyzők</h4>
        <a href="f1.php">Nyitólap</a>
        <ul>
        <!– BEGIN list_item –>
          <li><a
href="f1.php?number={NUMBER}">{NAME}</a>
        <!– END list_item –>
        </ul>
      <!– END list –>

    <!– BEGIN footer –>
      <hr>
      Látogató IP-je: {IP}
    <!– END  footer –>
  </body>
</html>
```

A PHP kód

```
<?php
       //az adatforrás
  $drivers[1] =
array('firstname'=>"Michael",'lastname'=>
➥"Schumacher",'number'=>1,'age'=>35,
➥'num_of_championships'=>7);
  $drivers[2] = array('firstname'=>"Mika",
➥'lastname'=>"Hakkinen",'number'=>
➥2,'age'=>33,'num_of_championships'=>2);
  $drivers[3] = array('firstname'=>"Jacques",
➥'lastname'=>"Villeneuve",'number'=>
➥3,'age'=>32,'num_of_championships'=>1);
  $drivers[4] = array('firstname'=>"Damon",
➥'lastname'=>"Hill",'number'=>4,
➥'age'=>37,'num_of_championships'=>1);

  require_once "HTML/Template/IT.php";
  $tpl = new HTML_Template_IT("./templates");
  $tpl->loadTemplatefile("main.tpl.html");

  $tpl->setCurrentBlock("header");
  $tpl->setVariable("DATE",date("Y.m.d"));
  $tpl->parseCurrentBlock();

  $tpl->setCurrentBlock("list_item");
  foreach ($drivers as $driver) {
    $tpl->setVariable("NUMBER",$driver
    ➥['number']);
    $tpl->setVariable("NAME",$driver
    ➥['firstname'][0].". ".$driver
    ➥['lastname']);
    $tpl->parseCurrentBlock();
  }
  $tpl->parse("list");

  $tpl->setCurrentBlock("footer");
  $tpl-
>setVariable("IP",$_SERVER['REMOTE_ADDR']);
  $tpl->parseCurrentBlock();

  $tpl->show();
?>
```

3. ábra Az eredmény

nálhatjuk, amely egy karaktersorozat formájában visszatérési értékéül adja a kitöltött sablont.

Még mindig tipegünk

A blokkok kitöltögetését természetesen egymás után végezhetjük, a dolgunk annyi, hogy meg kell mondani a megjelenítés előtt, hogy melyik blokk legyen a következő, amelyen műveleteket szeretnénk végezni. Egészítsük ki a fenti példát egy lábléccel, amely a látogató *IP*-jét mutatja!

Ismétlődő elemek, egymásba ágyazva

Ugorjunk most egy kicsit nagyobbat. Készítsünk egy adatforrásból egy listát a már meglévő lapon, amely a későbbiekben megjelenítendő világbajnokok neveit tartalmazza, amelyek mindegyi-

4. lista Blokkok

A sablon

```
<html>
  <body>
    <!- BEGIN header ->
      Üdvözöllek! Ma {DATE}. van.
      <hr>
    <!- END  header ->

    <table>
      <tr valign="top">
      <td>
      <!- BEGIN list ->
        <h4>Versenyzők</h4>
        <a href="f1.php">Nyitólap</a>
        <ul>
        <!- BEGIN list_item ->
          <li><a
href="f1.php?number={NUMBER}">{NAME}</a>
        <!- END list_item ->
        </ul>
      <!- END list ->
      </td>
      <td>

      <h4>Részletes adatok</h4>
      <!- BEGIN havetoselect ->
        Kérem, válasszon a felsorolt nevek közül.
      <!- END havetoselect ->

      <!- BEGIN detail ->
        Sorszám: {NUMBER}.<br>
        Vezetéknév: {LASTNAME}<br>
        Keresztnév: {FIRSTNAME}<br>
        Életkor: {AGE} év<br>
        Világbajnoki címek: {NUM_OF_CHAMPIONSHIPS}
        ➥db<br>
      <!- END detail ->
      </td>
      </tr>
    </table>

    <!- BEGIN footer ->
      <hr>
      Látogató IP-je: {IP}
    <!- END  footer ->
  </body>
</html>
```

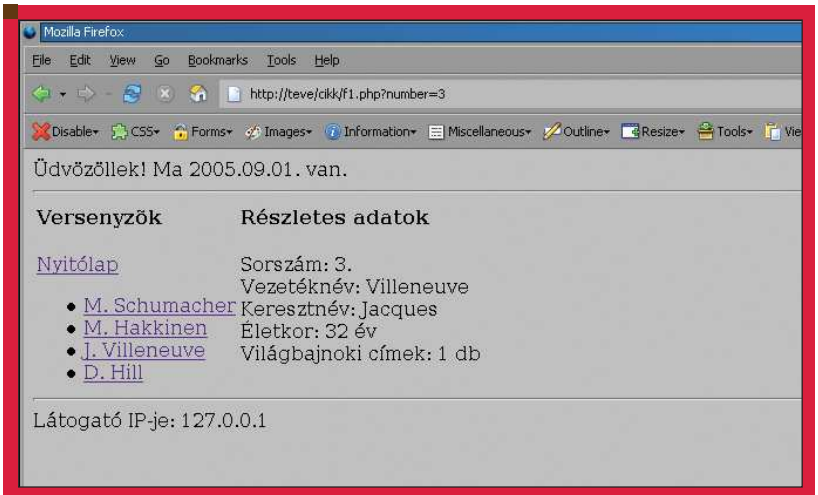
A PHP kód (az adatforrás nélkül)

```
<?php
        //az adatforrás: lást előző példa
 require_once "HTML/Template/IT.php";
  $tpl = new HTML_Template_IT("./templates");
  $tpl->loadTemplatefile("main.tpl.html");

  $tpl->setCurrentBlock("header");
  $tpl->setVariable("DATE",date("Y.m.d"));
  $tpl->parseCurrentBlock();

  $tpl->setCurrentBlock("list_item");
  foreach ($drivers as $driver) {
    $tpl->setVariable("NUMBER",$driver['number']);
    $tpl->setVariable("NAME",$driver['firstname']
    ➥[0].". ".$driver['lastname']);
    $tpl->parseCurrentBlock();
  }
  $tpl->parse("list");

  if (!isset($_GET[number])) {
     $tpl->touchBlock("havetoselect");
  } else {
    $tpl->setCurrentBlock("detail");
    $tpl->setVariable("FIRSTNAME",$drivers
    ➥[$_GET[number]]['firstname']);
    $tpl->setVariable("LASTNAME",$drivers[$_GET
    ➥[number]]['lastname']);
    $tpl->setVariable("NUMBER",$drivers
    ➥[$_GET[number]]['number']);
    $tpl->setVariable("AGE",$drivers
    ➥[$_GET[number]]['age']);
    $tpl->setVariable("NUM_OF_CHAMPIONSHIPS",
    ➥$drivers[$_GET[number]]
    ➥['num_of_championships']);

    $tpl->parseCurrentBlock();
  }

  $tpl->setCurrentBlock("footer");
  $tpl->setVariable("IP",$_SERVER['REMOTE_ADDR']);
  $tpl->parseCurrentBlock();

  $tpl->show();
?>
```

ke egy-egy hivatkozás, ami ***GET*** paraméterként tartalmazza a versenyzők sorszámát, valamint egy fix hivatkozást, amely az alapállapotba (nyitólapra) mutat.

A sablonban két egymásba ágyazott blokkot találunk. Általánosan igaz az, hogy mindig a legbelső blokk kitöltésével kell kezdenünk. Ez a fenti példában a listaelem (`list_item`) blokk. Már említettem, hogy egy-egy blokkot tetszőlegesen sokszor lefordíthatunk, és hozzárendelhetünk a kimenethez, így hozva létre ismétlődő ***HTML*** kódrészleteket. Jelen esetben egy ciklussal végigmentünk az adatforráson, miután kijelöltük a blokkot, majd a címke -> érték hozzárendelés után „hozzáfűztük” a lefordított sablonhoz, majd ezt ismételgettük ugyanezzel a blokkal addig, amíg el nem értük a kívánt számú listaelemet.

A listaelemek felszaporítása után „megparancsoltuk” a külső lista blokknak, hogy rendelje hozzá önmagát



4. ábra Így néz ki az elkészült példafeladat

a belső blokkal együtt a kitöltött kimenethez. Ha ez a külső blokk egyéb behelyettesítendő címkét is tartalmazott volna, akkor először meg kellett volna csinálni a hozzárendeléseket, majd utána utasítani a sablonkezelőt, hogy az immáron kész blokkot fűzze hozzá a végleges állapotúakhoz. Hasonlóan kell eljárni a tetszőleges számú egymásba ágyazott blokkok esetén is, folyamatosan belülről kifelé haladva.

Blokkok: akarom, nem akarom?

Egészítsük ki a példánkat a versenyzők részletes adatainak megjelenésével. Ha egy névre kattint a felhasználó, akkor a paraméterként kapott sorszám alapján a versenyzők listája mellet, jobbra jelenjen meg a pilóta összes adata. Ha még nincs kiválasztva egyetlen versenyző sem, akkor a részletes adatok helyett egy figyelmeztetés jelenjen meg, amelyben felszólítjuk a látogatót, hogy jelöljön meg egy nevet.
Már említettem, hogy alapértelmezetten azok a blokkok, amelyekkel nem végzünk műveleteket, nem is jelennek meg a kimenetben. Jelen esetben ezt a tulajdonságot használtuk ki, amikor a paraméter meglététől függően vagy az egyik, vagy a másik blokkot töltöttük ki. Illetve a dolog nem ilyen egyszerű, ugyanis a figyelmeztető blokk nem tartalmazott címkét, csak állandó szöveget, így azzal nem is tudunk műveleteket végezni. Ennek kezelésére való a `touchBlock()` metódus, amelynek hatására a megadott blokk változatlan formában bekerül a kész blokkok közé.
Mivel a fenti példában a sablon kezd elbonyolódni, célszerű lenne az egyes részeket külön fájlba tenni – már csak azért is, mivel a fejléc és a lábléc több oldalon is szerepelhet. Az *IT* az ilyen esetek kezelésében is jártas. Ha például a blokkok mentén vágnánk a fájlt külön darabokra, a fenti példában a kódot csak annyiban kellene módosítani, hogy a blokkok kijelölése előtt (`setCurrentBlock()`) be kellene tölteni még az adott nevű sablont (`loadTemplateFile`) is, majd ha a sablon összes címkéjét kicseréltük, a kimenetre írjuk azt, vagy a már emlegetett `get()` metódus segítségével egy változóban összefűzzük.

Végszó az IT és a sablonkezelés ügyében: egyszerűség!

Az előző bekezdés alapján már látszik, hogy a több fájlos megoldás (tortaszeletelő módszer) és ama bizonyos `get()` tagfüggvény segítségével, komoly, modulos rendszerű, komplett webes rendszerek készíthetők, úgy hogy mindvégig megőrizhetjük a módszer nemes egyszerűségét.
Ha jobban belegondolunk, ez a rendszer nem sok mindent tud: kicserél bizonyos címkéket bizonyos értékekre, amit már az `str_replace()` függvénnyel is megtehetünk. „Csupán” annyival fűszerezi a dolgot, hogy egy szemtelenül egyszerű utasításkészlettel burkolja a cserélgetést, illetve a blokkok alkalmazásával eléri, hogy kellően áttekinthető sablonokat készíthessünk. Ahhoz, hogy megértsük, mit csinál, elég elolvasni egy ehhez hasonló cikket. Ahhoz, hogy hatékonyan tudjunk vele dolgozni, nem kell további tetemes gyakorlati ismeretet szereznünk, egy saját magunk által kitalált kis példaalkalmazást megvalósítva könnyen ráérezhetünk az ízére.
Az elkészült kód egyszerű, áttekinthető, logikus és mások által is érthető – azt is mondhatnánk: közkincs. Az ilyen egyszerű megoldásokban rejlik a *PEAR* ereje!
Az *IT*-n kívül természetesen ott a többi sablonkezelő rendszer is, ám ennek a cikknek és a cikksorozatnak nem az a célja, hogy referenciát adjon a programfejlesztéshez. Sokkal inkább cél az, hogy bemutassa a *PEAR* rendszer erejét egy-egy jellegzetes témakörön keresztül. Aki a maradék *PEAR* sablonkezelőkről is szeretne részletesen tájékozódni, az alábbi oldalakon teheti ezt meg.

Komáromi Zoltán
(komi@kiskapu.hu)
25 éves, a BME hallgatója, mellette PHP-programozóként dolgozik. Kedvenc területe a multimédia.

KAPCSOLÓDÓ CÍMEK

Flexy:
➲ http://pear.php.net/package/HTML_Template_Flexy

PHPLIB:
➲ http://pear.php.net/package/HTML_Template_PHPLIB

Sigma:
➲ http://pear.php.net/package/HTML_Template_Sigma

Xipe:
➲ http://pear.php.net/package/HTML_Template_Xipe